Embarrassingly Parallel Benchmark Under PVM

David Browning
Computer Sciences Corporation
NASA Ames Research Center
Moffett Field, CA 94035-1000, USA

Abstract

Parallel Virtual Machine (PVM) is tested on a network of SGI workstations. A PVM version of the Embarrassingly Parallel (EP) benchmark, one of the NAS Parallel Benchmarks, is ported from an Intel iPSC/860 version. Details of the port and the author's experience with PVM are described. Performance results from both iPSC/860 and PVM versions of the EP benchmark are presented. PVM performed satisfactorily in these limited tests. Speedups can be poor under PVM because of additional, unexpected system loads on the workstations. Load balancing is shown to be a more important issue for PVM than for the Intel iPSC/860.

Introduction

Parallel Virtual Machine (PVM) [3] is a software package that allows one to treat a heterogeneous network of computers as a single parallel programming resource. The PVM package is comprised of two main parts: a daemon process run on each participating computer in the network, and a library of routines to access the PVM system from within a program. PVM was recently installed on the NAS Processing System Network.

This report describes an effort to test PVM on a simple parallel programming problem: the Embarrassingly Parallel (EP) benchmark, one of the benchmark kernels from the NAS Parallel Benchmarks [1]. The NAS Parallel Benchmarks are a collection of "paper and pencil" benchmarks -- so called because they are specified algorithmically in [1], and actual implementation is left up to the benchmarker. A set of sample codes demonstrating the benchmarks is available to assist in this implementation. The EP benchmark is computationally intensive and requires almost no communication between processors. This makes it a good choice as an initial test case for PVM.

EP Benchmark

E. Barszcz (Code RNR) provided sample Fortran programs for each of the NAS Parallel Benchmarks, including a version of the EP benchmark for the Intel iPSC/860 [2]. This particular code was written by P. Frederickson (formerly of RIACS) and D. Bailey (Code RNR). As provided, the sample code solves a smaller problem than required by [1]. Both problem size and memory requirements were easily modified through a single PARAMETER statement in the Fortran program. (The problem size is a measure of the total computational work required, not the amount of memory required.) The sample program of the EP benchmark prints the maximum elapsed time required by each of the processors executing the kernel.

To verify the correctness of the sample program, it was scaled up to the full size problem and run on the iPSC/860. Results from this test are shown in Figure 1 and Table 1.

To use PVM, the PVM daemon process (pvmd) is invoked on one of the computers in the network, typically the user's workstation. Pvmd reads a file listing other participating host computers and automatically invokes pvmd on each of those machines. Once pvmd is running on each host computer, the user executes a host program from the UNIX shell. This program will contain calls to PVM library routines, and will typically initiate additional node processes. The PVM system automatically executes these processes on the other host computers. Pvmd may be run in an interactive mode to monitor and control some aspects of the system.

The PVM library includes routines for initiation and synchronization of processes, and communication between processes. The PVM library is written in C, but a library of Fortran wrapper routines are provided so that most PVM library routines are available to Fortran programs. One exception is the routine pvm_mstat(), which returns an array of string pointers (C data type **char). This data type cannot be accessed from the Fortran language. The library routines in the PVM system provide for automatic conversion of binary data between host machines of different types.

As a matter of convenience in this effort, a single program was written to provide both host and node processes. The host process shares the computational workload equally with the node processes, but the host process provides additional control over the node processes and communicates with the user through standard input and output.

The PVM daemon process, the PVM library and the network of hosts together comprise a single parallel programming environment. To run the EP benchmark under PVM, it was necessary to modify parts of the sample code specific to the iPSC/860, including initialization, communication and timing, and port it to PVM. The relevant iPSC/860 commands and routines are listed in Table 2, along with PVM counterparts if any exist. Each is discussed more thoroughly below.

<u>Cubeexec</u>

On the iPSC/860 at NAS, the command

```
cubeexec -t8 nodeprog
```

performs the following functions:

1. Allocates and attaches eight node processors for the user;
2. Loads a copy of the node program *nodeprog* into each of the eight processors, which then begin execution;
3. Waits for all processors to finish execution;
4. Detaches the processors, returning them to the system.

Once compiled, the EP sample code comprises such a node program. Under PVM, the host program invoked by the user must perform these tasks. In the PVM version of the EP benchmark, the host program queries the user for the number of nodes desired, and initiates those processes with calls to the PVM library routine initiate(). Upon completion, each process must check out of PVM with a call to the PVM library routine leave(). Failure to do so will confuse the PVM daemon process.

Mynode(), Numnodes()

The iPSC/860 calls mynode() and numnodes() have no corresponding PVM library calls. However, each program running under PVM calls the PVM library routine enroll(), which returns an integer, starting from 0, uniquely identifying the PVM process. This integer can be saved and used in place of a call to mynode(). In lieu of a numnodes() call, the PVM host process must broadcast the total number of processes to each node process. A PVM node process has no way to determine this number itself.

Gdsum(), Gdhigh()

Of the many global arithmetic routines available to a node program on the iPSC/860, only two were required by the EP benchmark sample code: gdsum() and gdhigh(). Gdsum sums each element in an array of double precision floating point values across all the nodes. The array of global sums is then available in each node. Gdhigh operates similarly, returning the global maximum of each element instead of the sum. Presumably, such routines can be highly optimized on the iPSC/860 to take full advantage of the communication hardware and to avoid bottlenecks. In the PVM version of the EP benchmark, these routines were replaced with versions that worked under PVM. In each routine, the node processes send their data to the host, where the arithmetic is performed (sum or max), and then the host broadcasts the result to all the nodes.

Dclock()

The EP benchmark sample code used the iPSC/860 routine dclock() to perform timing calculations. Dclock() returns the elapsed time in seconds of each node. The maximum of these times is printed by the benchmark program. Under PVM, each node process is a UNIX process, so three times were of interest: user CPU time, user + system CPU time, and elapsed time. These were measured using BSD interval timers, and accessed via the BSD calls setitimer() and getitimer(). The PVM version was instrumented much more thoroughly than the sample code. Five sections of the benchmark were timed separately: initialization, including enrollment, initiation and the broadcast of numnodes; computation; and global summation of results with gdsum. (Gdhigh is not formally part of the benchmark. It is only used to determine the highest elapsed time of all the nodes, and is not timed itself.) The resulting timings are shown in Tables 4 and 5.

PVM

For the most part, PVM worked as described by its documentation. There were some discrepancies and ambiguities in the documentation, but all were minor. They were reported to the PVM developers using the email address supplied in the documentation. In one case, a PVM developer responded by supplying revised man pages via email the next day [4]. This demonstrates a high level of interest and support from the developers, and is a favorable feature of this package.

The user interface of pvmd had some minor deficiencies that made it somewhat inconvenient to use. None of them were serious enough to prevent one from using PVM effectively. Although pvmd may be run in the background, in which case it has no user interface, it was necessary to run pvmd interactively during development and testing of the EP benchmark, in order to control and monitor the progress of PVM processes. Unfortunately, it is not possible to execute a PVM host program directly from within an interactive pvmd session, making an additional shell window necessary when running pvmd interactively.

When invoked, pvmd reads a list of host computers from a file. PVM processes, whether invoked by the user or initiated by the host process, are assigned to the host computers in this list in a round-robin fashion. This is indeed the case when pvmd is first invoked, but the assignment of hosts is not as predictable in subsequent epochs. (A new epoch begins whenever there are no processes currently enrolled in the PVM system. At each new epoch, the sequential numbers returned by enroll() begin again at zero.) In one case, a host file listed four host computers. When pvmd was invoked with this hostfile, and the host process initiated three other processes, each of the four PVM processes was assigned to a distinct host computer in the list. Once the processes completed execution, a new epoch began in pvmd. (If any process fails to check out with a call to leave(), one may type "reset" in an interactive pvmd session to begin a new epoch.) When the same four PVM processes were initiated again, they were not assigned to the same four distinct host computers. Instead, one machine did not participate, and two processes were assigned to one of the other machines. This behavior was repeated in the next subsequent epoch, but with a different trio of host machines. For timing purposes, it was necessary to have the PVM processes assigned to distinct machines when executing the EP benchmark. Therefore it was necessary to quit and restart the pvmd session each time the benchmark was run. It was also necessary to quit and restart pvmd if the host file was modified, since it is not possible to reread the host file from an active pvmd session.

The user interface of pvmd would be improved if the following remedies were made:

1. It should be possible to execute a PVM host program directly from within an interactive pvmd session.
2. In subsequent epochs, PVM processes should be assigned to host computers in the same round robin fashion as in the first epoch.
3. It should be possible to reread the host file from within an interactive pvmd session without having to quit and restart pvmd.

Workstations

In order to limit the scope of this effort and to expedite its completion, only one type of computer was used in the PVM network: Silicon Graphics Inc. (SGI) IRIS workstations. This made it possible to compile a single binary executable file on the author's own SGI workstation and distribute it to other SGI workstations without modification. This simplified the development and testing of the benchmark under PVM. Table 3 lists the workstations that were used in the tests, their IP addresses, and their important hardware characteristics. All workstations had 16MBytes of main memory, and all ran the IRIX 4.0.1 operating system. As in the iPSC/860 version, each node process requires about 5 MBytes of memory, well below the 16 MBytes present on each workstation.

Results

The PVM version of the EP benchmark ran successfully on 1, 2 and 4 SGI workstations. Execution times and speedups are presented in Tables 4 and 5. Table 4 shows the results from the first tests, in which two of the workstations, wk77 and wk107, had additional, unexpected system loads. This caused wk77 to be a computational bottleneck in the 4-processor case, and wk107 in the 2-processor case. The speedups shown in Table 4 are considerably less than the speedups shown in Tables 1 and 5 for this reason. Upon further investigation, it was found that wk77 had a superfluous, runaway process contending for CPU time, leaving only half of the CPU for the PVM process. Wk107 had two similar runaway processes, leaving the PVM process with only one third of the CPU (albeit one third of a faster CPU). A second set of tests were performed after the runaway processes were killed. Results from these tests are shown in Table 5.

Conclusion

PVM performed satisfactorily in the limited tests described here. Some improvements were suggested for the pvmd user interface. The speedups shown in Table 4 are considerably lower than expected, but this is not due to the communication overhead within PVM. Instead, proper load balancing is shown to be a critical issue when resource availability is not known a priori. Because the EP benchmark is so computationally intensive and requires almost no communication, dynamic load balancing could be implemented fairly easily, and would effectively reduce the bottlenecks incurred from varying system loads. Such an effort is beyond the scope of this report. In other applications, where communication is a more important consideration, load balancing may be a far greater challenge.

```
BENCHMARK RESULTS:

CPU TIME =  141.9341

N = 2^   28

NO. GAUSSIAN PAIRS =     210832767.

COUNTS:

  0      98257395.

  1      93827014.

  2      17611549.

  3       1110028.

  4         26536.

  5           245.

  6             0.
  7             0.

  8             0.

  9             0.
```

Figure 1. EP benchmark sample output (iPSC/860 sample code).

Number Processors	Time (seconds)	Speedup
32	141.93	
64	71.00	2.00
128	35.56	3.99

Table 1. EP benchmark timings (iPSC/860 sample code).
Code compiled with `if77 -O2 -Knoieee`.
Speedups shown relative to 32-processor case.

iPSC/860	PVM Equivalent	Remarks

cubeexec	initiate()
	enroll(), leave()
		Performed by host process.
		Performed by each process.

mynode()	N/A	Use return value of enroll().
numnodes() gdsum	N/A N/A	Broadcast by host process. Custom routine required.
gdhigh	N/A	Custom routine required.

Table 2. iPSC/860 *vs.* PVM: Commands and library routines.

Hostname	IP Address	Hardware Characteristics
wk40	129.99.48.3	4D/25TG, 20 MHz IP6 processor
wk77	129.99.48.7	4D/25TG, 20 MHz IP6 processor
wk84 willard (wk96)[*]	129.99.50.26 129.99.48.61	4D/25TG, 20 MHz IP6 processor 4D/25TG, 20 MHz IP6 processor
wk107	129.99.48.6	4D/35TG, 36 MHz IP12 processor

Table 3. Workstations.
(*) Author's workstation.

Hostname/ Mynode	Times (Seconds)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
willard/	User	0.00	0.00	0.00	9791.50	0.00	9791.50	
0	User+Sys	0.04	0.00	0.01	9800.03	0.01	9800.09	
	Elapsed	50.09	0.04	0.01	9898.46	0.22	9948.82	1
willard/	User	0.01	0.00	0.00	4895.29	0.00	4895.30	
0	User+Sys	0.06	0.00	0.00	4900.45	0.01	4900.52	
	Elapsed	50.13	0.03	0.01	4957.46	619.30	5626.93	1.77
wk107/	User	0.01	0.00	0.01	2335.81	0.00	2335.83	
1	User+Sys	0.01	0.00	0.01	2340.13	0.00	2340.11	
	Elapsed	10.06	0.00	0.01	5576.36	0.00	5586.43	
willard/	User	0.00	0.00	0.00	2448.89	0.00	2448.89	
0	User+Sys	0.05	0.03	0.00	2452.50	0.04	2452.62	
	Elapsed	50.24	0.18	0.00	2483.14	1470.31	4003.87	2.48
wk40/	User	0.00	0.00	0.00	2440.18	0.01	2440.19	
1	User+Sys	0.04	0.00	0.00	2442.01	0.02	2442.07	
	Elapsed	10.08	0.00	0.08	2456.60	1496.95	3963.71	
wk77/	User	0.00	0.00	0.00	2446.15	0.00	2446.15	
2	User+Sys	0.03	0.00	0.00	2449.01	0.01	2449.05	
	Elapsed	40.07	0.00	0.04	3953.53	0.13	3993.77	
wk107	User	0.01	0.00	0.00	1166.28	0.00	1166.29	
3	User+Sys	0.01	0.00	0.00	1169.59	0.01	1169.61	
	Elapsed	10.03	0.00	0.03	2630.10	1323.22	3963.38	

Table 4. EP benchmark under PVM using 1, 2 and 4 workstations.
Additional system load on wk77 and wk107.

Key to column entries:

(1) Enroll() library call.
(2) Initiation of node processes by host process. Includes reading *stdin* to determine number of processes to initiate. Does not include delay between host's initiate() call and node's startup.
(3) Broadcast (by host) and receipt (by nodes) of numnodes.
(4) Floating point computation.
(5) PVM version of gdsum() routine.
(6) Total of (1)-(5). Max elapsed time used in speedup calculation (7).
(7) Speedup over 1-workstation case. Adjacent to figures used in calcu-

lation.

Hostname/ Mynode	Times (Seconds)	(1)	(2)	(3)	(4)	(5)	(6)	(7)
Willard/	User	0.00	0.00	0.00	9799.17	0.00	9799.17	
0	User+Sys	0.04	0.00	0.01	9812.75	0.01	9812.81	
	Elapsed	50.08	0.03	0.01	10007.28	0.16	10057.56	1
Willard/	User	0.00	0.00	0.00	4901.33	0.00	4901.33	
0	User+Sys	0.02	0.01	0.00	4905.41	0.00	4905.44	
	Elapsed	50.11	0.07	0.01	4963.40	0.08	5013.67	2.01
wk40/	User	0.00	0.00	0.00	4877.19	0.00	4877.19	
1	User+Sys	0.02	0.00	0.00	4880.84	0.00	4880.86	
	Elapsed	10.06	0.00	0.00	4900.45	63.13	4973.64	
Willard/	User	0.02	0.00	0.00	2449.14	0.00	2449.16	
0	User+Sys	0.05	0.01	0.00	2452.69	0.00	2452.75	
	Elapsed	50.15	0.20	0.02	2485.79	0.24	2536.40	3.97
wk40/	User	0.00	0.00	0.00	2438.72	0.00	2438.72	
1	User+Sys	0.03	0.00	0.00	2440.61	0.03	2440.67	
	Elapsed	10.06	0.00	0.00	2450.78	35.26	2496.09	
wk77/	User	0.00	0.00	0.00	2439.27	0.00	2439.27	
2	User+Sys	0.01	0.00	0.01	2441.48	0.01	2441.51	
	Elapsed	40.04	0.00	0.03	2451.89	34.22	2526.18	
wk84	User	0.01	0.00	0.00	2453.35	0.00	2453.36	
3	User+Sys	0.02	0.00	0.00	2469.62	0.01	2469.65	
	Elapsed	20.04	0.00	0.01	2477.52	8.55	2506.12	

Table 5. EP benchmark under PVM using 1, 2 and 4 workstations.
No significant additional system load.

Key to column entries:

(1) Enroll() library call.
(2) Initiation of node processes by host process. Includes reading *stdin* to determine number of processes to initiate. Does not include delay between host's initiate() call and node's startup.
(3) Broadcast (by host) and receipt (by nodes) of numnodes.
(4) Floating point computation.
(5) PVM version of gdsum() routine.
(6) Total of (1)-(5). Max elapsed time used in speedup calculation (7).
(7) Speedup over 1-workstation case. Adjacent to figures used in calcu-

lation.

References

[1] D. H. Bailey, J. Barton, T. Lasinski, H. Simon, eds., "The NAS Parallel Benchmarks," NAS Report RNR-91-002, January 1991.

[2] E. Barszcz, NAS Applied Research Branch, personal communication, June 1992.

[3] A. Beguelin, J. Dongarra, A. Geist, R. Manchek, V. Sunderam, "A User's Guide to PVM: Parallel Virtual Machine," Report ORNL/TM-11826, Oak Ridge National Laboratory, Oak Ridge, TN, July 1991.

[4] R. Manchek, Computer Science Department, University of Tennessee, Knoxville, personal communication, June 1992.